Kayla Carr

11 February, 2016

Will Code For Food

**Metric: Modularity**

The encapsulation ratio:

|  |
| --- |
| Calculates the ratio of encapsulated classes to all classes for each package. A class is said to be encapsulated if it is not used externally to the package. In general, most of the classes in a package should not be used externally to the package, so this ratio should be fairly high. |

|  |  |
| --- | --- |
| **Package** | **Encapsulation Rate** |
| *Total Average* | *48.31%* |
| view | 100% |
| view.summermissions | 75% |
| view.ridesharing | 66.67% |
| view.resources | 80% |
| view.getinvolved | 0% |
| view.events | 60% |
| view.common | 76.47% |
| values | 20% |
| tasks | 25% |
| model.ridesharingTests | 100% |
| model.ridesharing | 25% |
| model.resources | 71.43% |
| model.getInvolved | 50% |
| model.formTests | 100% |
| model.common | 100% |
| model.common.messaging | 33.33% |
| model.common.list | 100% |
| model.common.form | 0% |
| model.common.common.users | 0% |
| model.common.common | 22.22% |
| controller | 0% |
| controller.retrieval | 25% |
| controller.api\_interfaces | 0% |

**Analysis**

A lot of the encapsulation ratios here are a bit misleading due to how we structured our project. The low modularity for values and tasks makes sense, since values are meant to be accessed by everyone (and should perhaps be moved to the common package to reflect this) and tasks are used by the view activities. Tasks relating to loading UI elements should thus probably be moved into the respective view package (i.e. events, summermissions, etc.). The controller package also has low modularity since we stored our retrievers in there, which are used by almost all features to retrieve information from the database (events, missions, rides, etc.). Common.common also contains utilities which were meant to be used by other classes.

It appears as if the views are, for the most part, successfully decoupled from each other. The getinvolved view package currently does not contain anything and will be ignored for now. Taking all this into account, our project appears fairly well encapsulated and follows a Model-View-Controller format. That being said, I still feel as if we could restructure some of our packages to make more sense in terms of features, rather than separating out the models from views and controllers. I believe this was the intention when Cru asked for a modularized project, that way they could take one package/feature from one project and insert it into another if necessary, without having to search through several files or link things up again.